Republic of the Philippines
CAMARINES SUR POLYTECHNIC COLLEGES
Nabua, Camarines Sur

COLLEGE *of* COMPUTER STUDIES

CCCS 104 - Data Structures and Algorithms

LEARNING TASK (LINEAR DATA STRUCTURE - LIST)

NAME: ***JAYSON ASIADO*** SECTION: ***BSCS 2A***

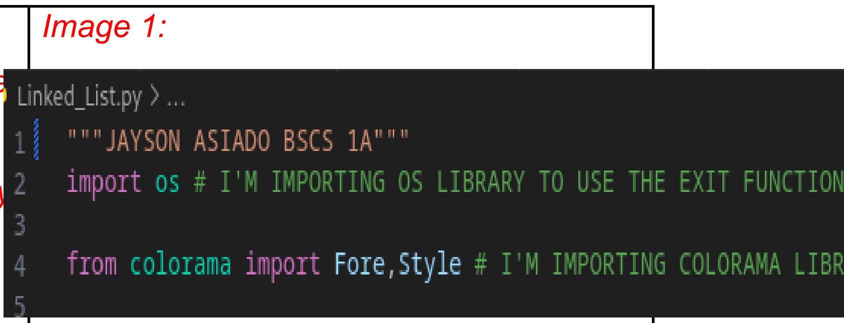
RATIONALE

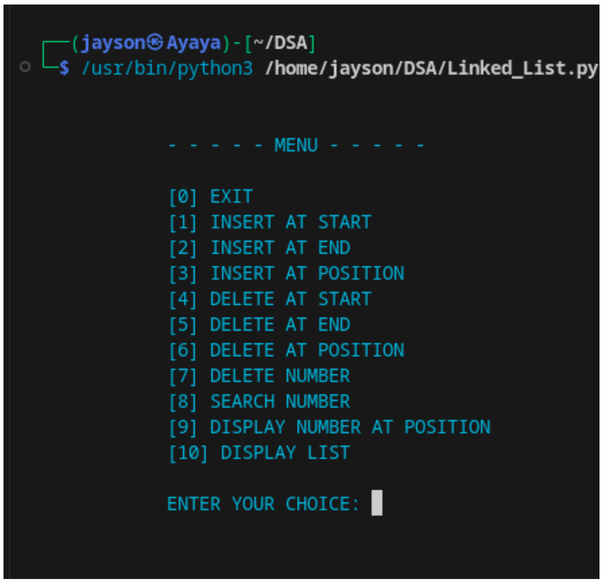
Linear Data structure is a type of data structure which is the elements ordered sequentially or linearly each element has its unique predecessor and successor except for the last and first element. It is ordered so you can traverse it one by one in a single dimension. Common examples of this are the Arrays, Stack, Linked list, Queue.

In this Learning task the given instructions is to implement Linked list so I write a Linked list code and add all the possible features of it. Example of the function of a linked list is add,delete,insert,search and it has head and tail nodes each element pointing to the next value except the tail which is the next element is null or None.

USER GUIDE

Step by step instructions on how to use your program. Include images for easily visualization

<table>
<tr>
<td><em>Step 1:</em>
<em>To run the program you need to install the library called Colorama by typing in the terminal ‘pip install colorama’ . This library I used has the ability to give simple color to the text, red for errors and green for successful execution.</em></td>
<td><em>Image 1:</em>

<pre>
"""JAYSON ASIADO BSCS 1A"""
import os # I'M IMPORTING OS LIBRARY TO USE THE EXIT FUNCTION

from colorama import Fore,Style # I'M IMPORTING COLORAMA LIBR
</pre></td>
</tr>
<tr>
<td><em>Step 2:</em>
<em>Just navigate the run button or in pycharm just press ‘shift + alt10’ to run then you will see the menu interface just like in the picture.</em></td>
<td>
<pre>
┌──(jayson㉿Ayaya)-[~/DSA]
└─$ /usr/bin/python3 /home/jayson/DSA/Linked_List.py

            - - - - - MENU - - - - -

            [0] EXIT
            [1] INSERT AT START
            [2] INSERT AT END
            [3] INSERT AT POSITION
            [4] DELETE AT START
            [5] DELETE AT END
            [6] DELETE AT POSITION
            [7] DELETE NUMBER
            [8] SEARCH NUMBER
            [9] DISPLAY NUMBER AT POSITION
            [10] DISPLAY LIST

            ENTER YOUR CHOICE:
</pre></td>
</tr>
</table>

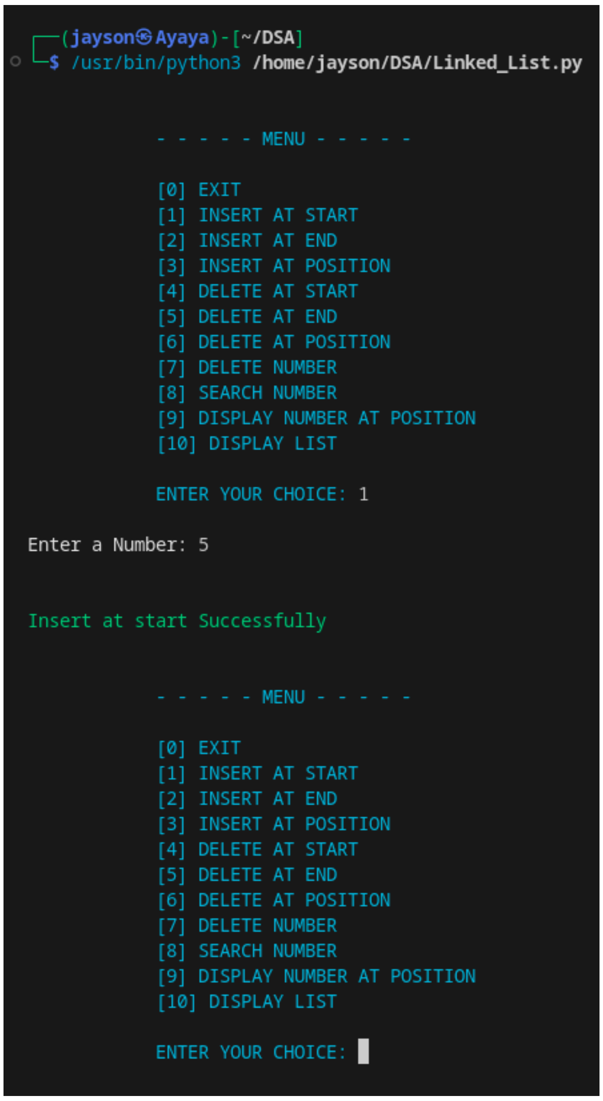
Step 3:

After you see the menu interface you can now enter your choice between 0 - 10 corresponds to each function.

First we will try the number 1 option because 0 is just exiting the program we will use that at the last step. now enter then enter you will see a “Enter a number” then type the number you want that input will accept only integer.

As you can see I just added the number 5 and it says that it is successfull.

Then the program will return to the menu, ready to accept input again

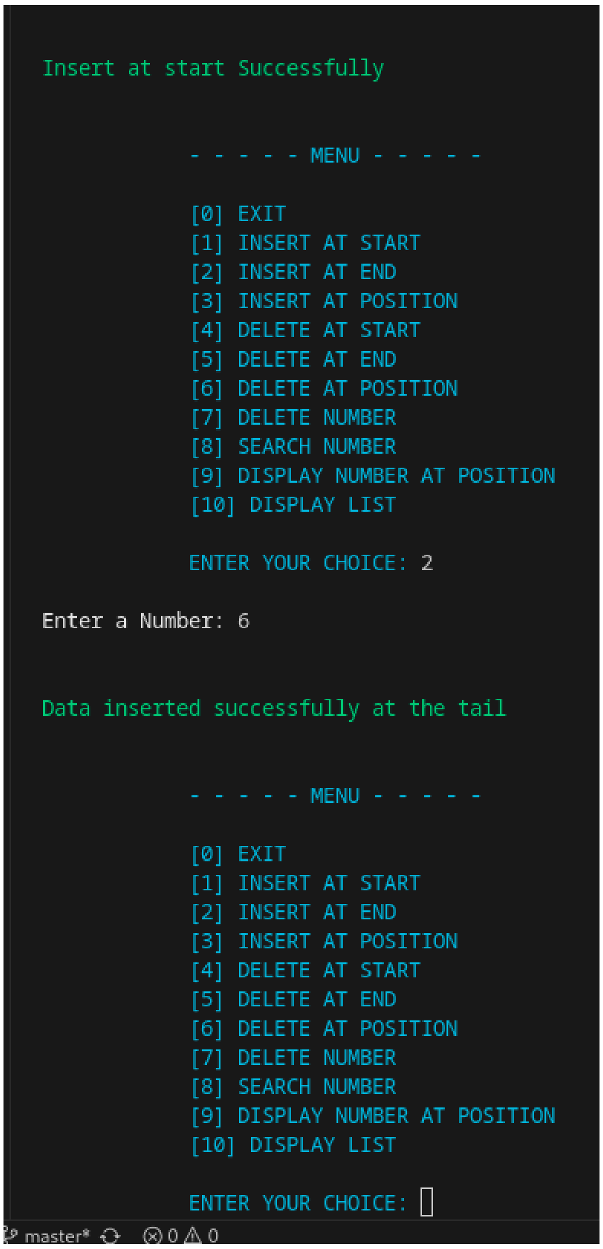


```
┌──(jayson㉿Ayaya)-[~/DSA]
└─$ /usr/bin/python3 /home/jayson/DSA/Linked_List.py

                - - - - - MENU - - - - -

                [0] EXIT
                [1] INSERT AT START
                [2] INSERT AT END
                [3] INSERT AT POSITION
                [4] DELETE AT START
                [5] DELETE AT END
                [6] DELETE AT POSITION
                [7] DELETE NUMBER
                [8] SEARCH NUMBER
                [9] DISPLAY NUMBER AT POSITION
                [10] DISPLAY LIST

                ENTER YOUR CHOICE: 1

Enter a Number: 5


Insert at start Successfully


                - - - - - MENU - - - - -

                [0] EXIT
                [1] INSERT AT START
                [2] INSERT AT END
                [3] INSERT AT POSITION
                [4] DELETE AT START
                [5] DELETE AT END
                [6] DELETE AT POSITION
                [7] DELETE NUMBER
                [8] SEARCH NUMBER
                [9] DISPLAY NUMBER AT POSITION
                [10] DISPLAY LIST

                ENTER YOUR CHOICE: █
```

Step 4:

Now lets try option 2, It will ask again for a number but unlike the option one, it will insert to the tail node. Also if the list is empty it will become the head because the head must be the first element in the linked list.

This time I added number 6 to the list. Like before, the program is ready again for inputs after the successful insert.



```
Insert at start Successfully


              - - - - - MENU - - - - -

              [0] EXIT
              [1] INSERT AT START
              [2] INSERT AT END
              [3] INSERT AT POSITION
              [4] DELETE AT START
              [5] DELETE AT END
              [6] DELETE AT POSITION
              [7] DELETE NUMBER
              [8] SEARCH NUMBER
              [9] DISPLAY NUMBER AT POSITION
              [10] DISPLAY LIST

              ENTER YOUR CHOICE: 2

Enter a Number: 6


Data inserted successfully at the tail


              - - - - - MENU - - - - -

              [0] EXIT
              [1] INSERT AT START
              [2] INSERT AT END
              [3] INSERT AT POSITION
              [4] DELETE AT START
              [5] DELETE AT END
              [6] DELETE AT POSITION
              [7] DELETE NUMBER
              [8] SEARCH NUMBER
              [9] DISPLAY NUMBER AT POSITION
              [10] DISPLAY LIST

              ENTER YOUR CHOICE: 
```

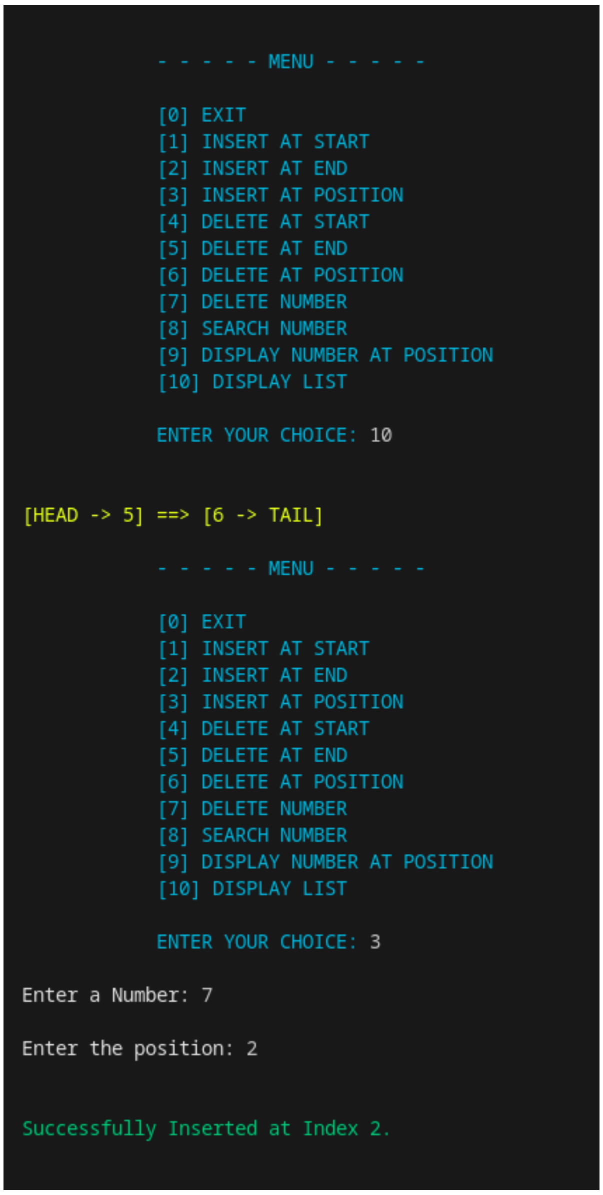
Step 5:

Now let's go to option 3, by the words itself means inserting at the given position but if you have an empty list you have only index 0 to be inserted.

Also I added a feature when you can also insert in the negative index not only in the positive index.

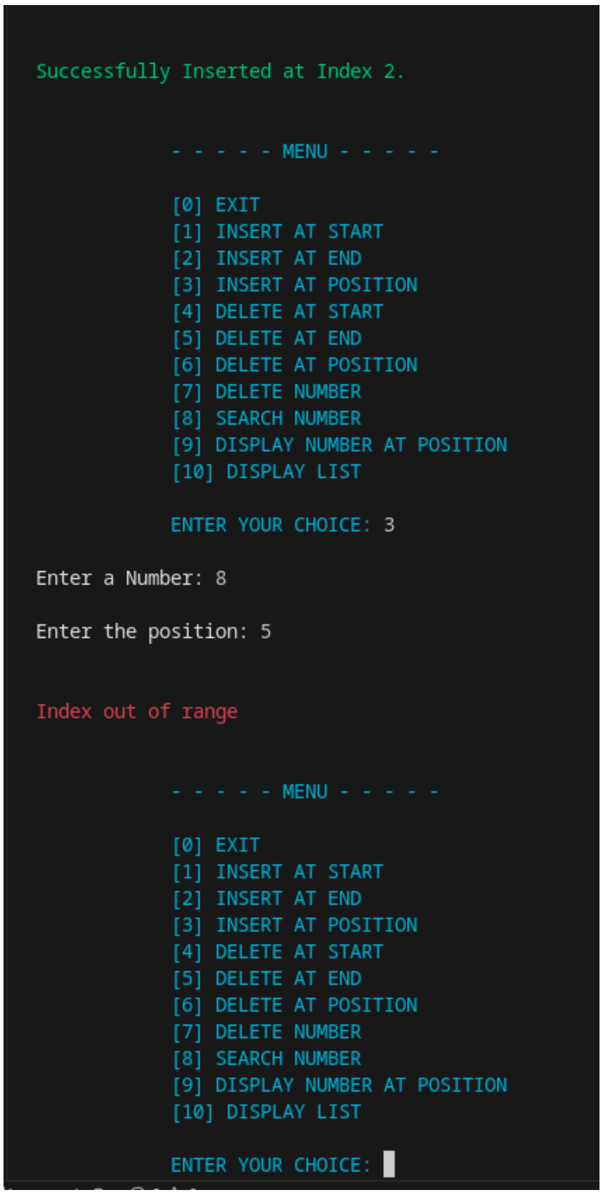
Previously we had a linked list with 2 values: the head -> 5 , 6 <- tail, to see the current list we can use option 10 to display the list.

Now let's try adding at index 2 since we have 0-3 available index to insert.



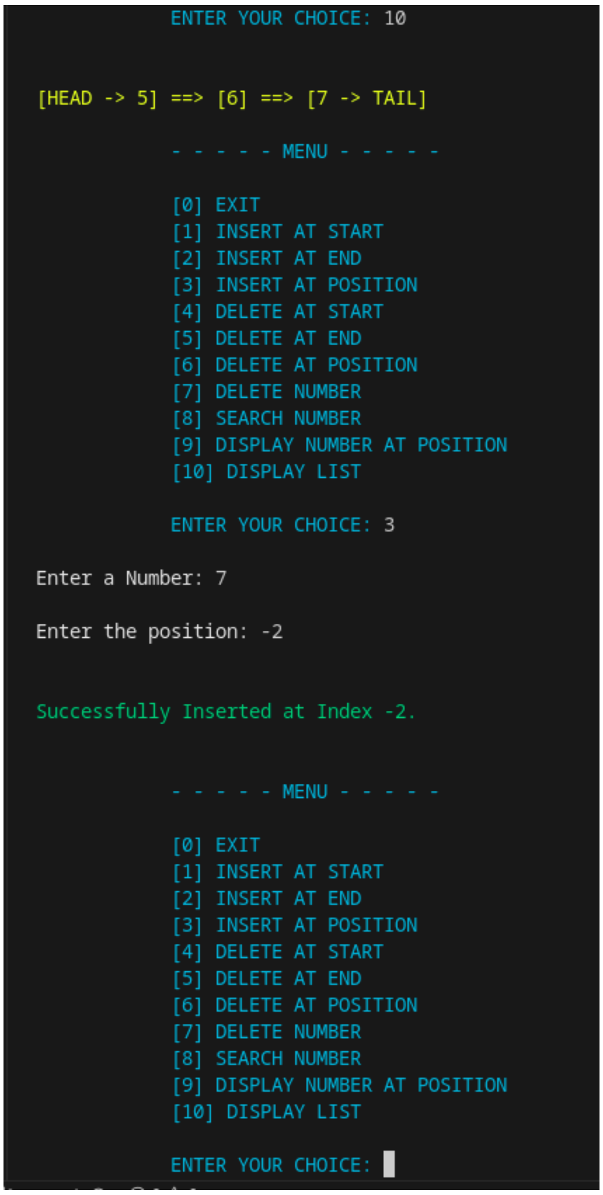
```
          - - - - - MENU - - - - -

          [0] EXIT
          [1] INSERT AT START
          [2] INSERT AT END
          [3] INSERT AT POSITION
          [4] DELETE AT START
          [5] DELETE AT END
          [6] DELETE AT POSITION
          [7] DELETE NUMBER
          [8] SEARCH NUMBER
          [9] DISPLAY NUMBER AT POSITION
          [10] DISPLAY LIST

          ENTER YOUR CHOICE: 10


[HEAD -> 5] ==> [6 -> TAIL]

          - - - - - MENU - - - - -

          [0] EXIT
          [1] INSERT AT START
          [2] INSERT AT END
          [3] INSERT AT POSITION
          [4] DELETE AT START
          [5] DELETE AT END
          [6] DELETE AT POSITION
          [7] DELETE NUMBER
          [8] SEARCH NUMBER
          [9] DISPLAY NUMBER AT POSITION
          [10] DISPLAY LIST

          ENTER YOUR CHOICE: 3

Enter a Number: 7

Enter the position: 2


Successfully Inserted at Index 2.
```

I'll also try to insert at index 5 to see the exception as you can see you cant add to the out of range index it'll also prompt if you enter an out of range negative index.



```
Successfully Inserted at Index 2.


            - - - - - MENU - - - - -

            [0] EXIT
            [1] INSERT AT START
            [2] INSERT AT END
            [3] INSERT AT POSITION
            [4] DELETE AT START
            [5] DELETE AT END
            [6] DELETE AT POSITION
            [7] DELETE NUMBER
            [8] SEARCH NUMBER
            [9] DISPLAY NUMBER AT POSITION
            [10] DISPLAY LIST

            ENTER YOUR CHOICE: 3

Enter a Number: 8

Enter the position: 5


Index out of range


            - - - - - MENU - - - - -

            [0] EXIT
            [1] INSERT AT START
            [2] INSERT AT END
            [3] INSERT AT POSITION
            [4] DELETE AT START
            [5] DELETE AT END
            [6] DELETE AT POSITION
            [7] DELETE NUMBER
            [8] SEARCH NUMBER
            [9] DISPLAY NUMBER AT POSITION
            [10] DISPLAY LIST

            ENTER YOUR CHOICE: 
```

Next I inserted a value of number 7 to the index -2 since it is not out of range. then it says Successfully inserted at the given index -2.

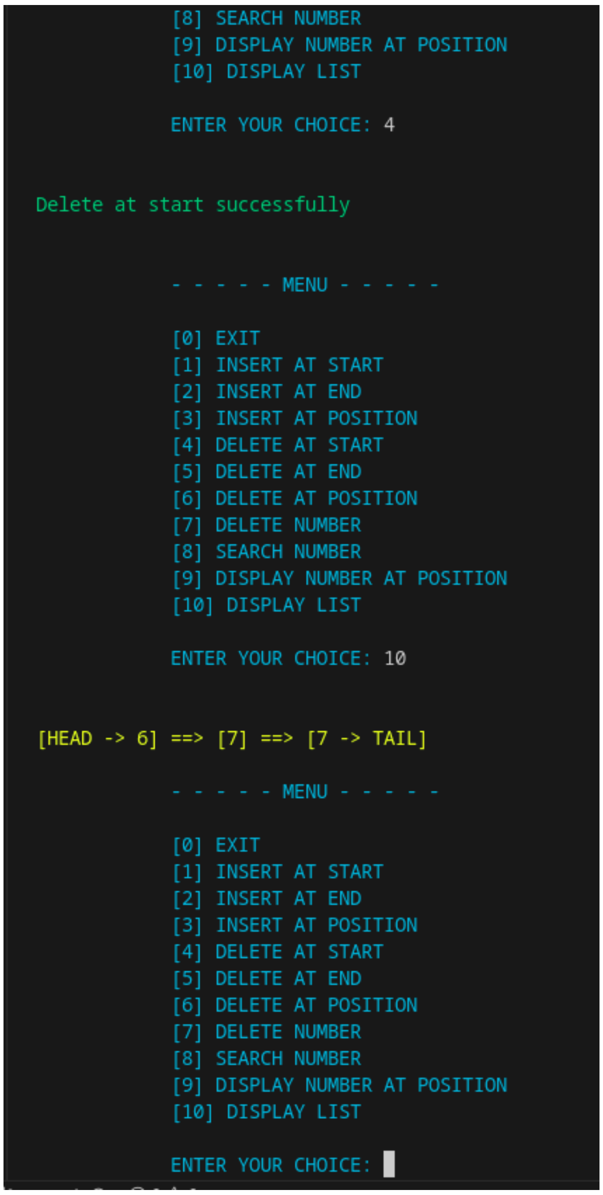


```
                ENTER YOUR CHOICE: 10


   [HEAD -> 5] ==> [6] ==> [7 -> TAIL]

                - - - - - MENU - - - - -

                [0] EXIT
                [1] INSERT AT START
                [2] INSERT AT END
                [3] INSERT AT POSITION
                [4] DELETE AT START
                [5] DELETE AT END
                [6] DELETE AT POSITION
                [7] DELETE NUMBER
                [8] SEARCH NUMBER
                [9] DISPLAY NUMBER AT POSITION
                [10] DISPLAY LIST

                ENTER YOUR CHOICE: 3

   Enter a Number: 7

   Enter the position: -2


   Successfully Inserted at Index -2.


                - - - - - MENU - - - - -

                [0] EXIT
                [1] INSERT AT START
                [2] INSERT AT END
                [3] INSERT AT POSITION
                [4] DELETE AT START
                [5] DELETE AT END
                [6] DELETE AT POSITION
                [7] DELETE NUMBER
                [8] SEARCH NUMBER
                [9] DISPLAY NUMBER AT POSITION
                [10] DISPLAY LIST

                ENTER YOUR CHOICE:
```

Step 6:

Option 4 is very specific, it will only delete the head node and the next of it is the new head.

I chose option 4 and as you can see after I display the updated linked list the value of head which is 5 is gone because it is already deleted.

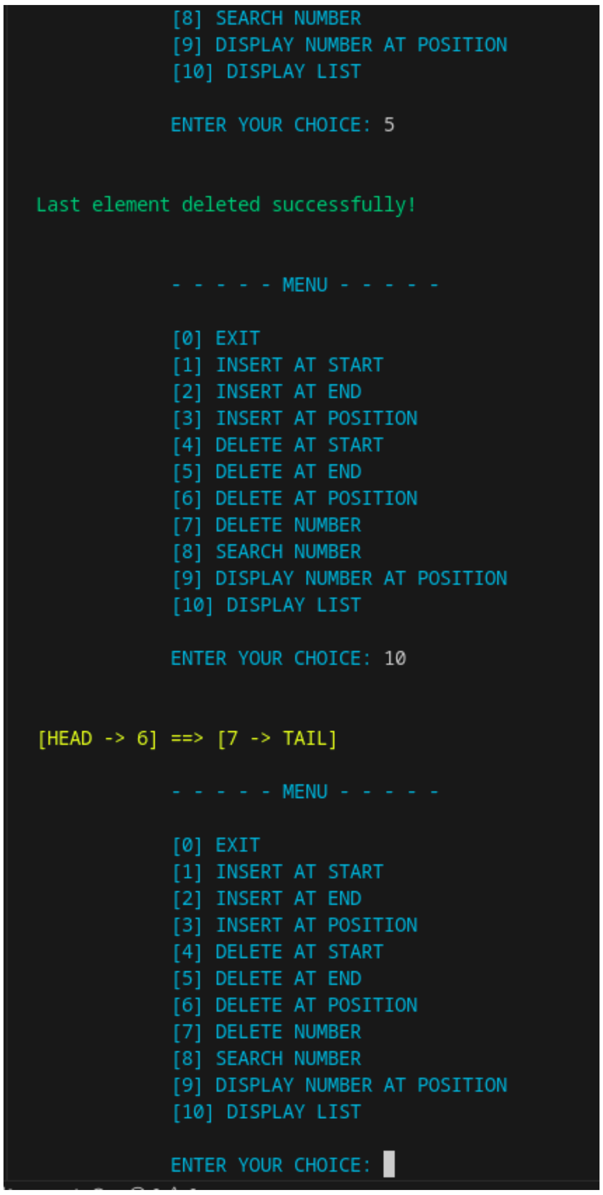


```
            [8] SEARCH NUMBER
            [9] DISPLAY NUMBER AT POSITION
            [10] DISPLAY LIST

            ENTER YOUR CHOICE: 4


  Delete at start successfully


            - - - - - MENU - - - - -

            [0] EXIT
            [1] INSERT AT START
            [2] INSERT AT END
            [3] INSERT AT POSITION
            [4] DELETE AT START
            [5] DELETE AT END
            [6] DELETE AT POSITION
            [7] DELETE NUMBER
            [8] SEARCH NUMBER
            [9] DISPLAY NUMBER AT POSITION
            [10] DISPLAY LIST

            ENTER YOUR CHOICE: 10


  [HEAD -> 6] ==> [7] ==> [7 -> TAIL]

            - - - - - MENU - - - - -

            [0] EXIT
            [1] INSERT AT START
            [2] INSERT AT END
            [3] INSERT AT POSITION
            [4] DELETE AT START
            [5] DELETE AT END
            [6] DELETE AT POSITION
            [7] DELETE NUMBER
            [8] SEARCH NUMBER
            [9] DISPLAY NUMBER AT POSITION
            [10] DISPLAY LIST

            ENTER YOUR CHOICE:
```

Step 7:

Option 5, same with the delete at start but the tail node is the one that will get deleted.

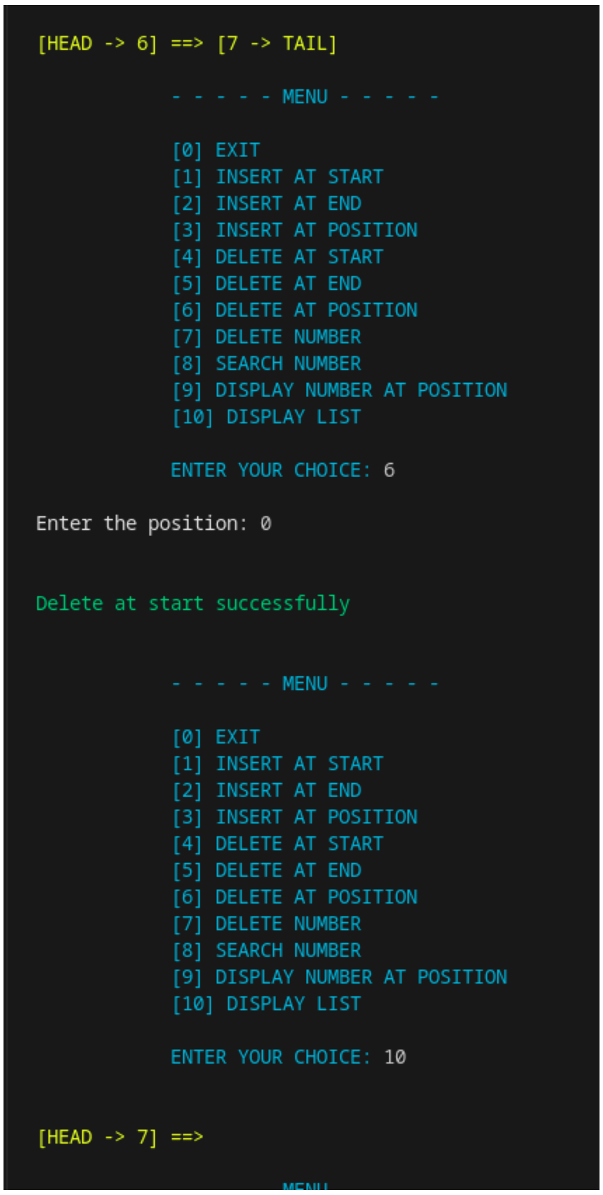
This is the updated list after option 5 is used.



```
             [8] SEARCH NUMBER
             [9] DISPLAY NUMBER AT POSITION
             [10] DISPLAY LIST

             ENTER YOUR CHOICE: 5


  Last element deleted successfully!


             - - - - - MENU - - - - -

             [0] EXIT
             [1] INSERT AT START
             [2] INSERT AT END
             [3] INSERT AT POSITION
             [4] DELETE AT START
             [5] DELETE AT END
             [6] DELETE AT POSITION
             [7] DELETE NUMBER
             [8] SEARCH NUMBER
             [9] DISPLAY NUMBER AT POSITION
             [10] DISPLAY LIST

             ENTER YOUR CHOICE: 10


  [HEAD -> 6] ==> [7 -> TAIL]

             - - - - - MENU - - - - -

             [0] EXIT
             [1] INSERT AT START
             [2] INSERT AT END
             [3] INSERT AT POSITION
             [4] DELETE AT START
             [5] DELETE AT END
             [6] DELETE AT POSITION
             [7] DELETE NUMBER
             [8] SEARCH NUMBER
             [9] DISPLAY NUMBER AT POSITION
             [10] DISPLAY LIST

             ENTER YOUR CHOICE:
```

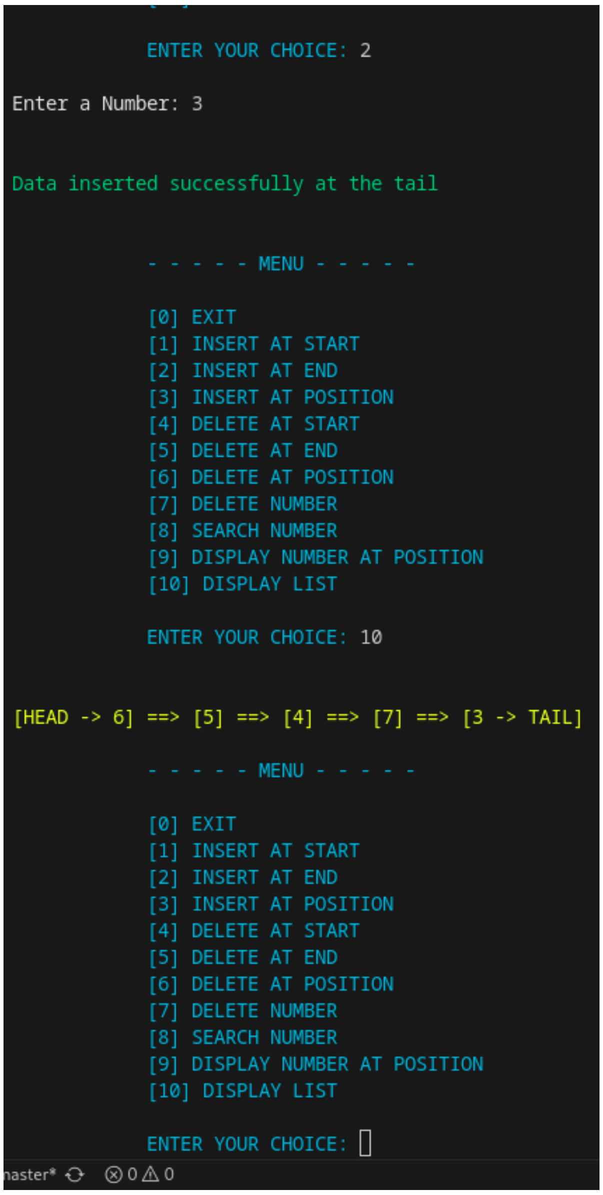
Step 8:

Option 6 is the same as option 3 but instead of insert, this one will delete the value in the given position.



```
[HEAD -> 6] ==> [7 -> TAIL]

            - - - - - MENU - - - - -

            [0] EXIT
            [1] INSERT AT START
            [2] INSERT AT END
            [3] INSERT AT POSITION
            [4] DELETE AT START
            [5] DELETE AT END
            [6] DELETE AT POSITION
            [7] DELETE NUMBER
            [8] SEARCH NUMBER
            [9] DISPLAY NUMBER AT POSITION
            [10] DISPLAY LIST

            ENTER YOUR CHOICE: 6

Enter the position: 0


Delete at start successfully


            - - - - - MENU - - - - -

            [0] EXIT
            [1] INSERT AT START
            [2] INSERT AT END
            [3] INSERT AT POSITION
            [4] DELETE AT START
            [5] DELETE AT END
            [6] DELETE AT POSITION
            [7] DELETE NUMBER
            [8] SEARCH NUMBER
            [9] DISPLAY NUMBER AT POSITION
            [10] DISPLAY LIST

            ENTER YOUR CHOICE: 10


[HEAD -> 7] ==>

                      MENU
```

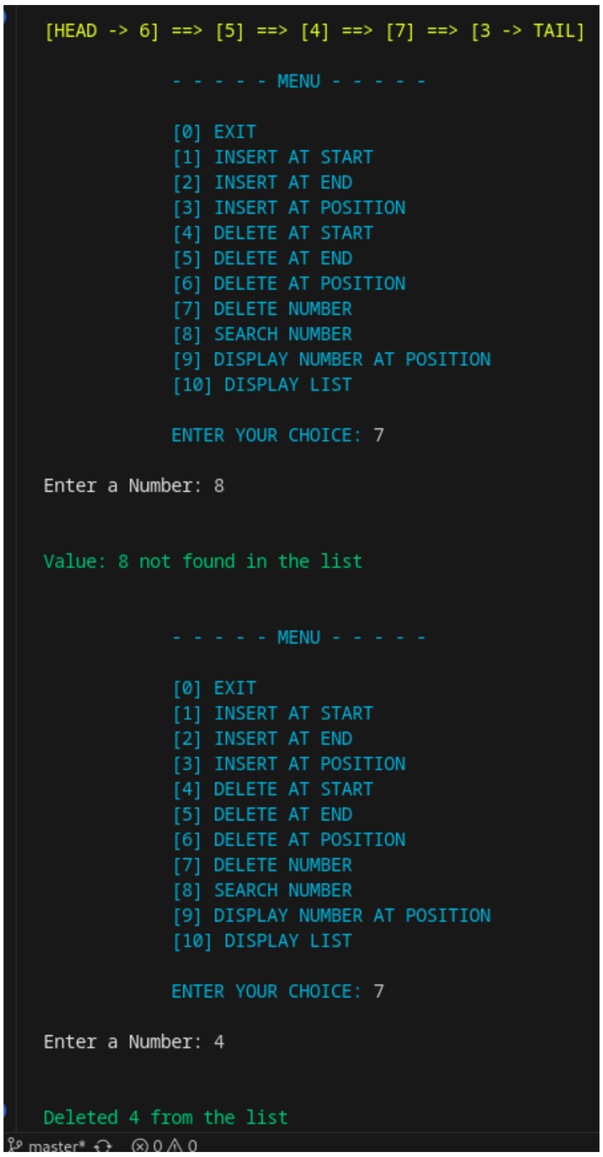
Now we only have one element in the linked list so I'll add some elements that we can use for the remaining options.



```
                ENTER YOUR CHOICE: 2

Enter a Number: 3


Data inserted successfully at the tail


                - - - - - MENU - - - - -

                [0] EXIT
                [1] INSERT AT START
                [2] INSERT AT END
                [3] INSERT AT POSITION
                [4] DELETE AT START
                [5] DELETE AT END
                [6] DELETE AT POSITION
                [7] DELETE NUMBER
                [8] SEARCH NUMBER
                [9] DISPLAY NUMBER AT POSITION
                [10] DISPLAY LIST

                ENTER YOUR CHOICE: 10


[HEAD -> 6] ==> [5] ==> [4] ==> [7] ==> [3 -> TAIL]

                - - - - - MENU - - - - -

                [0] EXIT
                [1] INSERT AT START
                [2] INSERT AT END
                [3] INSERT AT POSITION
                [4] DELETE AT START
                [5] DELETE AT END
                [6] DELETE AT POSITION
                [7] DELETE NUMBER
                [8] SEARCH NUMBER
                [9] DISPLAY NUMBER AT POSITION
                [10] DISPLAY LIST

                ENTER YOUR CHOICE: 
```

Step 9:

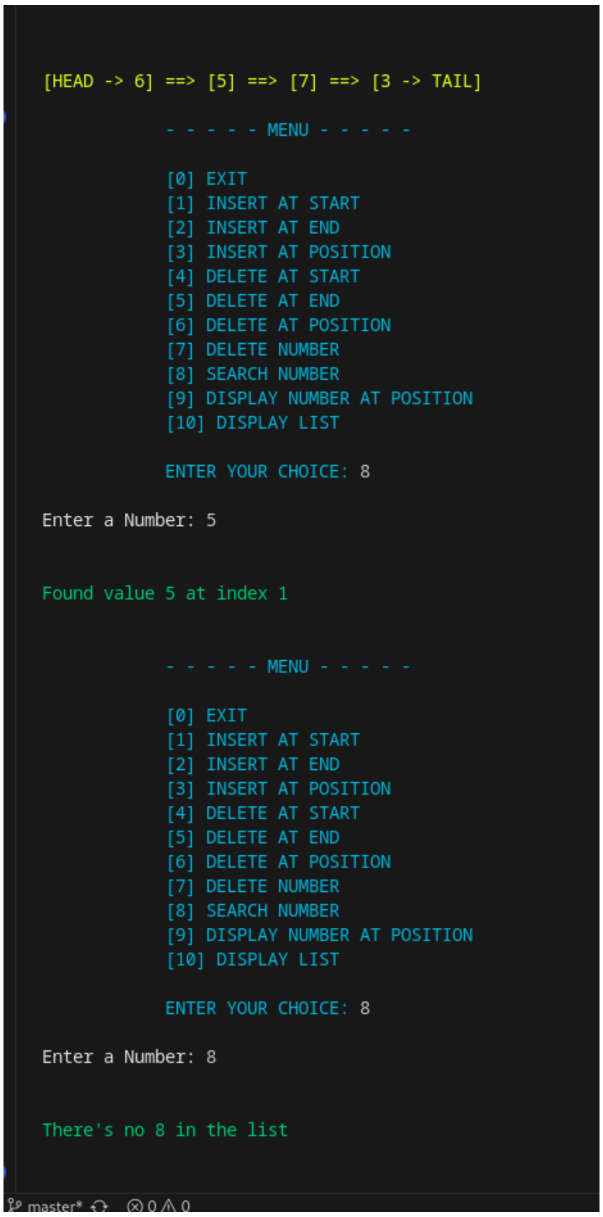
Option 7 will ask you for the value to delete and if that value exists in the list it will be deleted and if the number occurs more than one in the list it will only delete the first occurence of that value. If the value is not on the list after traversing it linearly . So to be able to understand this option I'll try to delete the number that doesn't exist and the one that exists to see the different prompt.



```
 [HEAD -> 6] ==> [5] ==> [4] ==> [7] ==> [3 -> TAIL]

             - - - - - MENU - - - - -

             [0] EXIT
             [1] INSERT AT START
             [2] INSERT AT END
             [3] INSERT AT POSITION
             [4] DELETE AT START
             [5] DELETE AT END
             [6] DELETE AT POSITION
             [7] DELETE NUMBER
             [8] SEARCH NUMBER
             [9] DISPLAY NUMBER AT POSITION
             [10] DISPLAY LIST

             ENTER YOUR CHOICE: 7

 Enter a Number: 8


 Value: 8 not found in the list


             - - - - - MENU - - - - -

             [0] EXIT
             [1] INSERT AT START
             [2] INSERT AT END
             [3] INSERT AT POSITION
             [4] DELETE AT START
             [5] DELETE AT END
             [6] DELETE AT POSITION
             [7] DELETE NUMBER
             [8] SEARCH NUMBER
             [9] DISPLAY NUMBER AT POSITION
             [10] DISPLAY LIST

             ENTER YOUR CHOICE: 7

 Enter a Number: 4


 Deleted 4 from the list
```

Step 10:

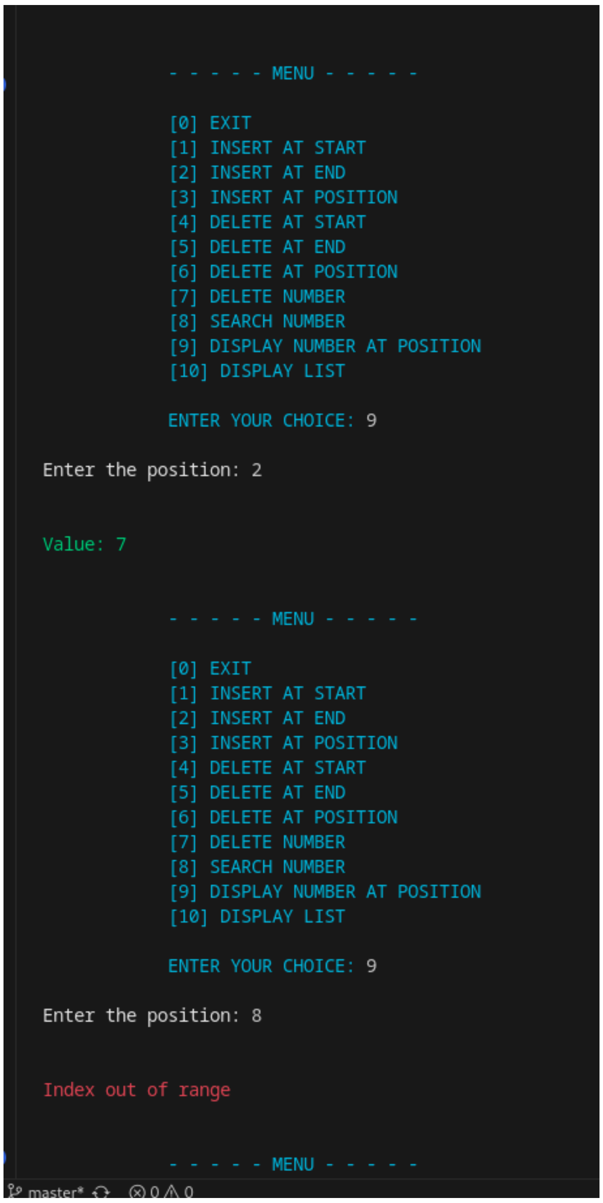
Option 8 will search through the list to find the number that is equal to your input and if it found that number it will return the index corresponding to that value and if not it will say that the number is not found in the list after searching.



```
[HEAD -> 6] ==> [5] ==> [7] ==> [3 -> TAIL]

            - - - - - MENU - - - - -

            [0] EXIT
            [1] INSERT AT START
            [2] INSERT AT END
            [3] INSERT AT POSITION
            [4] DELETE AT START
            [5] DELETE AT END
            [6] DELETE AT POSITION
            [7] DELETE NUMBER
            [8] SEARCH NUMBER
            [9] DISPLAY NUMBER AT POSITION
            [10] DISPLAY LIST

            ENTER YOUR CHOICE: 8

Enter a Number: 5


Found value 5 at index 1


            - - - - - MENU - - - - -

            [0] EXIT
            [1] INSERT AT START
            [2] INSERT AT END
            [3] INSERT AT POSITION
            [4] DELETE AT START
            [5] DELETE AT END
            [6] DELETE AT POSITION
            [7] DELETE NUMBER
            [8] SEARCH NUMBER
            [9] DISPLAY NUMBER AT POSITION
            [10] DISPLAY LIST

            ENTER YOUR CHOICE: 8

Enter a Number: 8


There's no 8 in the list
```

Step 11:

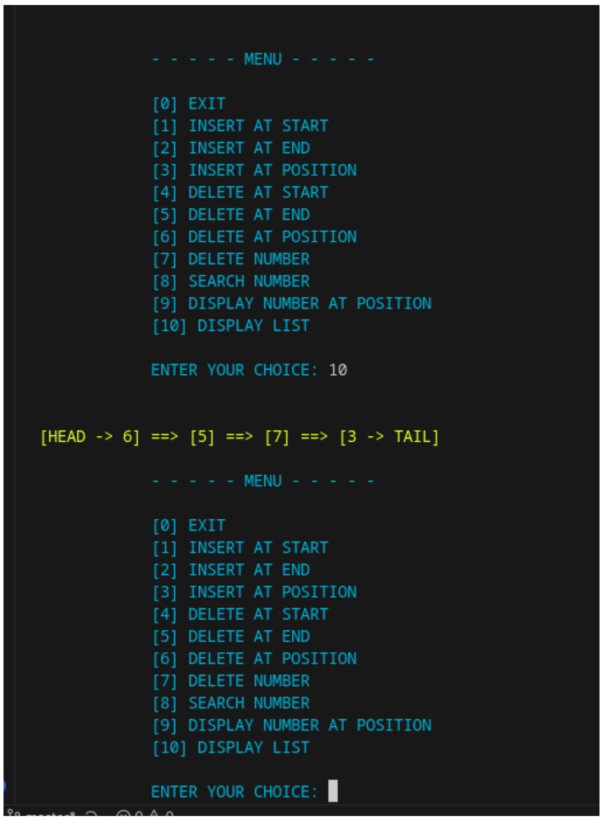
Option 9 will ask the index and if that index is not out of range in the list the it will return the value of the given index.



```
               - - - - - MENU - - - - -

               [0] EXIT
               [1] INSERT AT START
               [2] INSERT AT END
               [3] INSERT AT POSITION
               [4] DELETE AT START
               [5] DELETE AT END
               [6] DELETE AT POSITION
               [7] DELETE NUMBER
               [8] SEARCH NUMBER
               [9] DISPLAY NUMBER AT POSITION
               [10] DISPLAY LIST

               ENTER YOUR CHOICE: 9

Enter the position: 2


Value: 7


               - - - - - MENU - - - - -

               [0] EXIT
               [1] INSERT AT START
               [2] INSERT AT END
               [3] INSERT AT POSITION
               [4] DELETE AT START
               [5] DELETE AT END
               [6] DELETE AT POSITION
               [7] DELETE NUMBER
               [8] SEARCH NUMBER
               [9] DISPLAY NUMBER AT POSITION
               [10] DISPLAY LIST

               ENTER YOUR CHOICE: 9

Enter the position: 8


Index out of range


               - - - - - MENU - - - - -
```

Step 12:

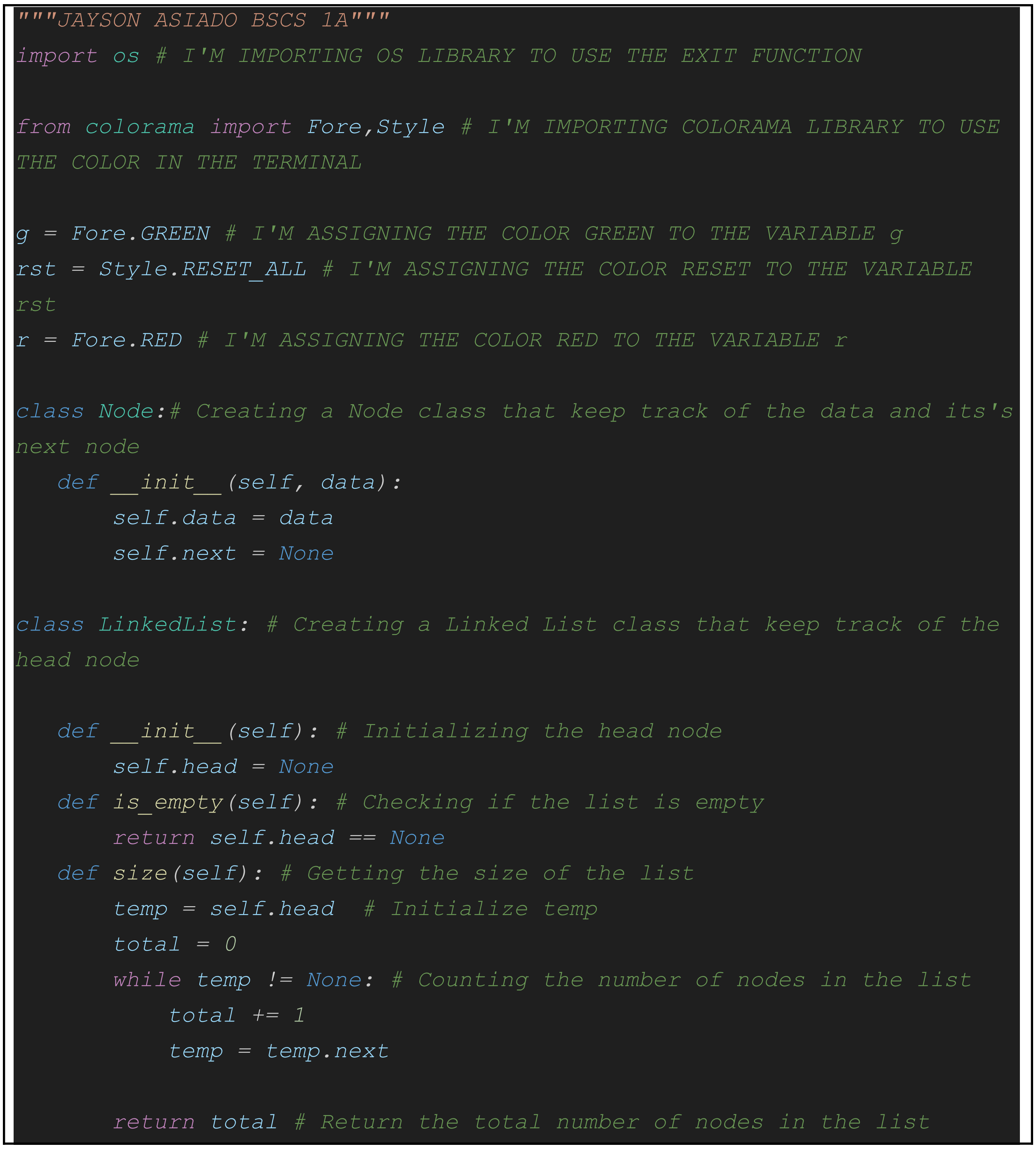
We already used it many times while using this program but I'll also explain what this does. The display function displays the linked list in order from the head to the tail and has an arrow pointing to its next value.



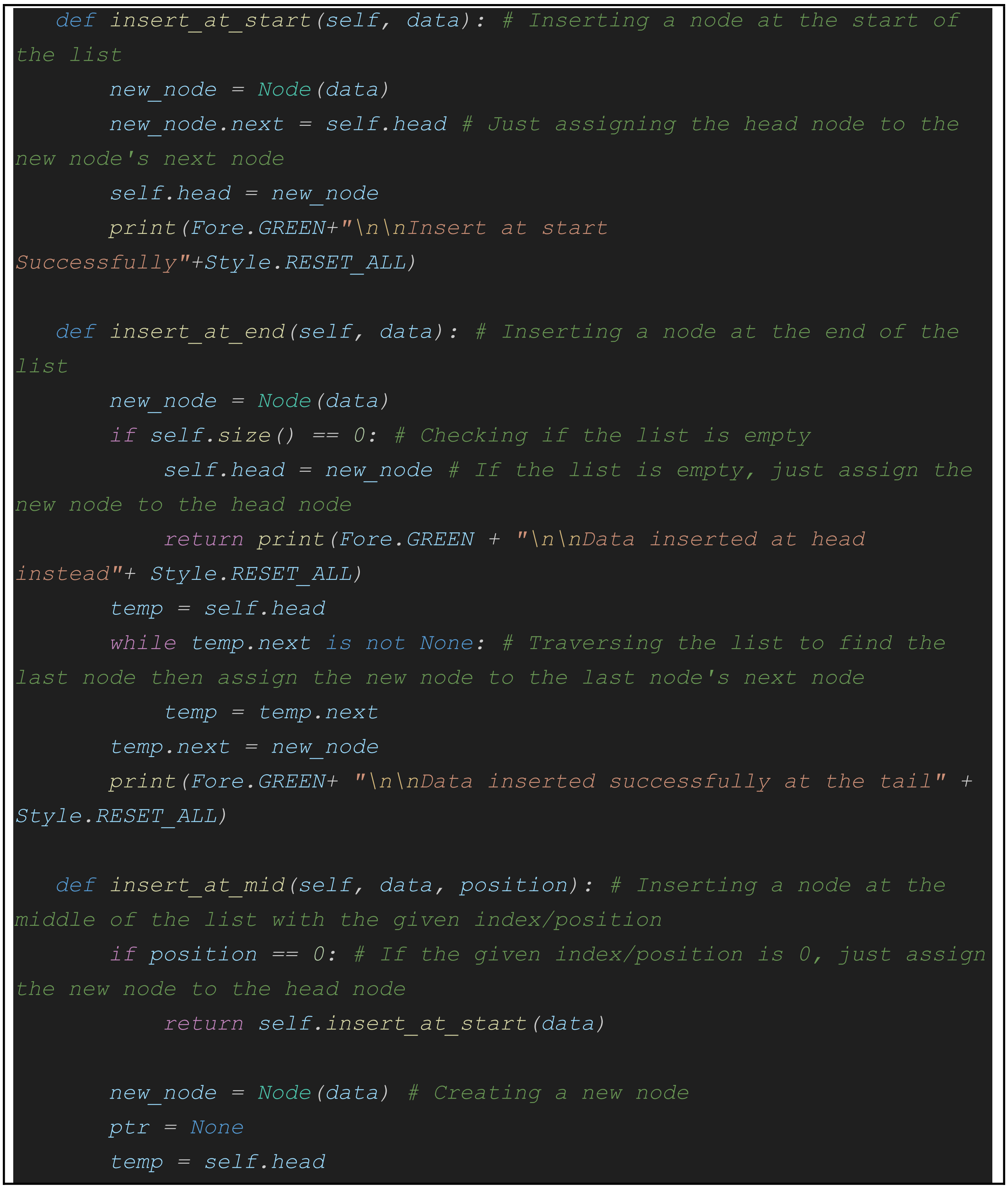
```
                - - - - - MENU - - - - -

                [0] EXIT
                [1] INSERT AT START
                [2] INSERT AT END
                [3] INSERT AT POSITION
                [4] DELETE AT START
                [5] DELETE AT END
                [6] DELETE AT POSITION
                [7] DELETE NUMBER
                [8] SEARCH NUMBER
                [9] DISPLAY NUMBER AT POSITION
                [10] DISPLAY LIST

                ENTER YOUR CHOICE: 10


    [HEAD -> 6] ==> [5] ==> [7] ==> [3 -> TAIL]

                - - - - - MENU - - - - -

                [0] EXIT
                [1] INSERT AT START
                [2] INSERT AT END
                [3] INSERT AT POSITION
                [4] DELETE AT START
                [5] DELETE AT END
                [6] DELETE AT POSITION
                [7] DELETE NUMBER
                [8] SEARCH NUMBER
                [9] DISPLAY NUMBER AT POSITION
                [10] DISPLAY LIST

                ENTER YOUR CHOICE: 
```

Step 13: *The last step is the option 0 which will prompt a message confirmation if you want to exit or not. If you choose ‘n’ the program will return to the menu interface. Otherwise it will exit the program saying “THANK YOU FOR USING THIS PROGRAM”*	

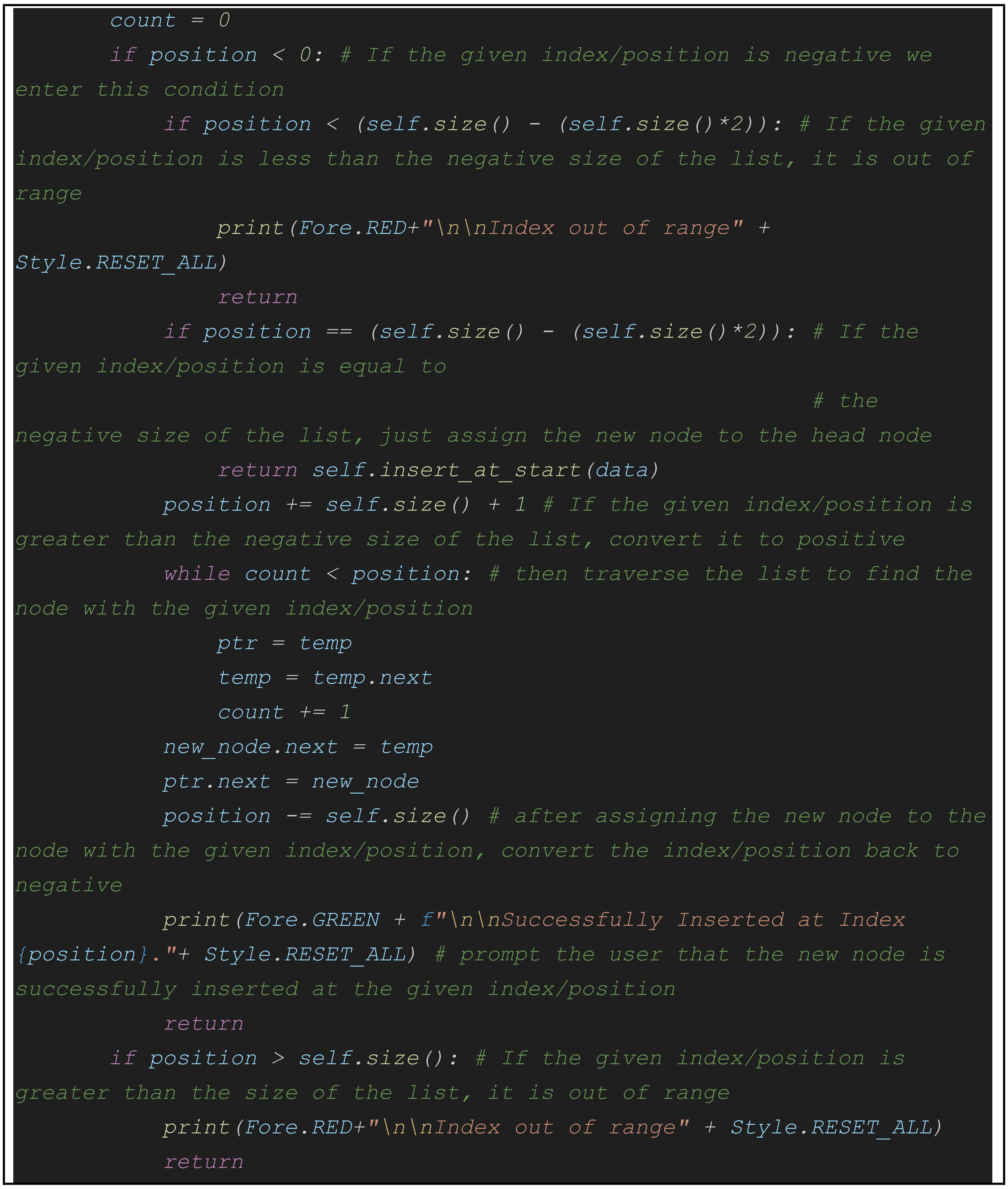
PROGRAM CODE



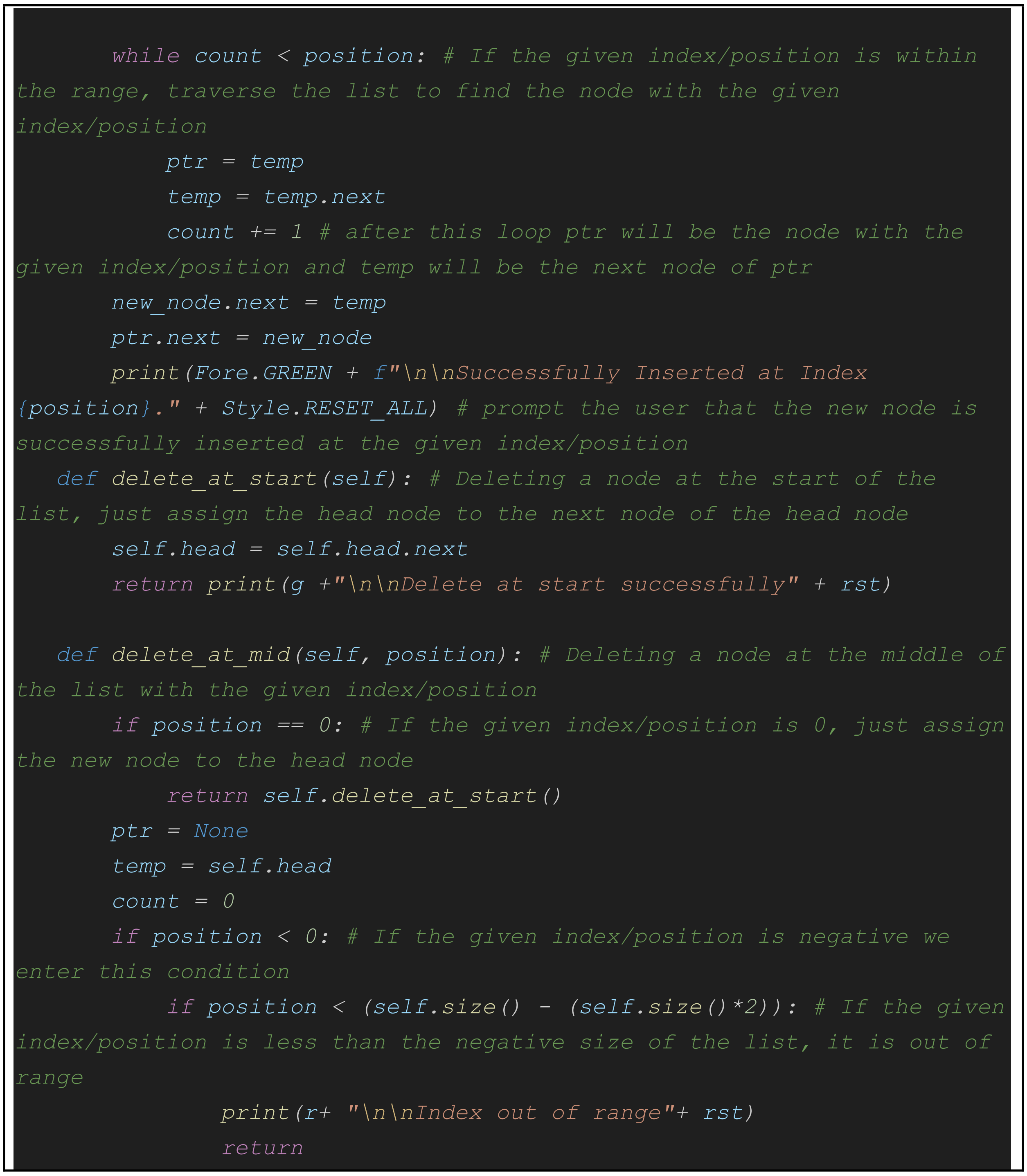
```
"""JAYSON ASIADO BSCS 1A"""
import os # I'M IMPORTING OS LIBRARY TO USE THE EXIT FUNCTION

from colorama import Fore,Style # I'M IMPORTING COLORAMA LIBRARY TO USE THE COLOR IN THE TERMINAL

g = Fore.GREEN # I'M ASSIGNING THE COLOR GREEN TO THE VARIABLE g
rst = Style.RESET_ALL # I'M ASSIGNING THE COLOR RESET TO THE VARIABLE rst
r = Fore.RED # I'M ASSIGNING THE COLOR RED TO THE VARIABLE r

class Node:# Creating a Node class that keep track of the data and its's next node
    def __init__(self, data):
        self.data = data
        self.next = None

class LinkedList: # Creating a Linked List class that keep track of the head node

    def __init__(self): # Initializing the head node
        self.head = None
    def is_empty(self): # Checking if the list is empty
        return self.head == None
    def size(self): # Getting the size of the list
        temp = self.head  # Initialize temp
        total = 0
        while temp != None: # Counting the number of nodes in the list
            total += 1
            temp = temp.next

        return total # Return the total number of nodes in the list
```



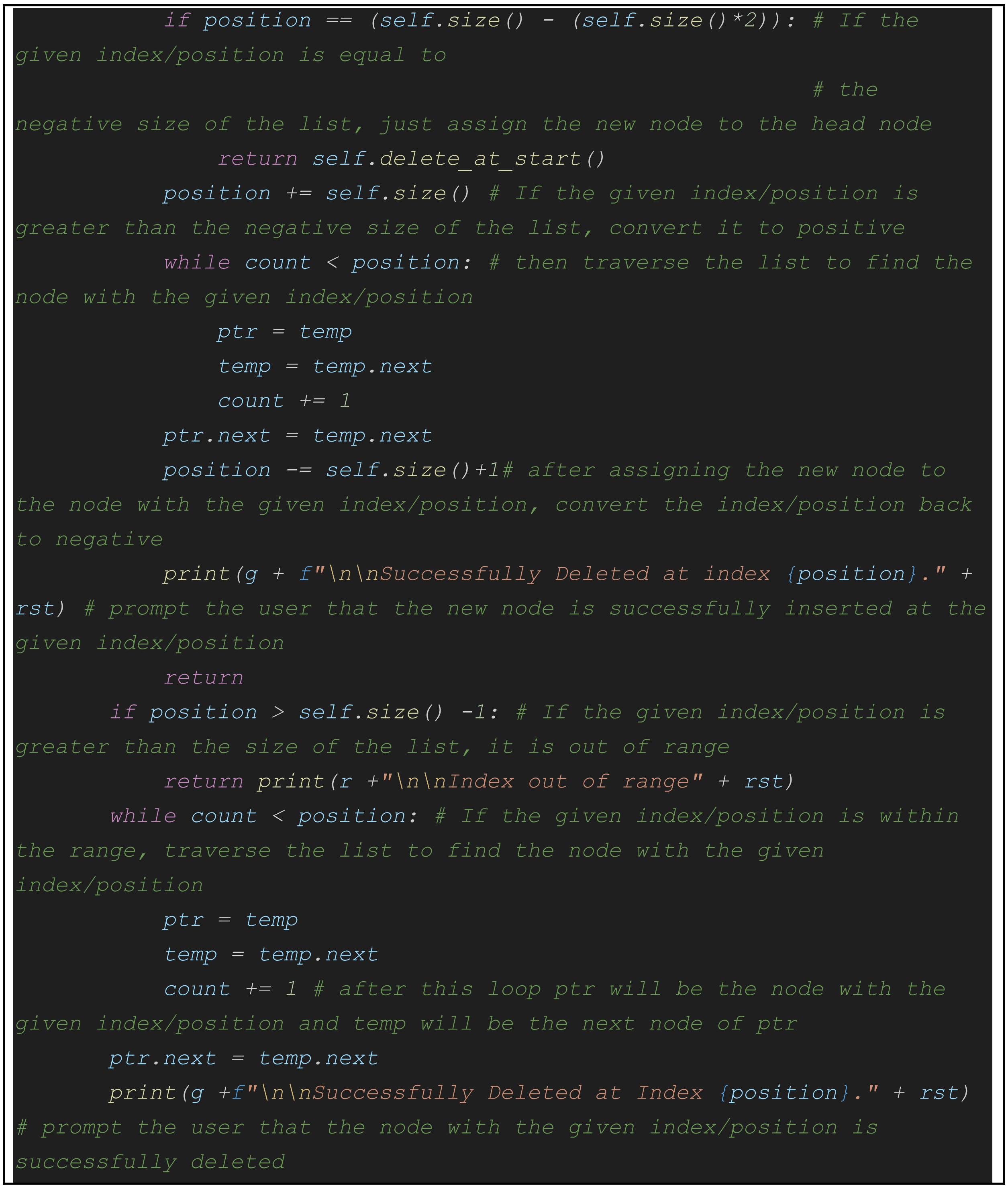
```
    def insert_at_start(self, data): # Inserting a node at the start of
the list
        new_node = Node(data)
        new_node.next = self.head # Just assigning the head node to the
new node's next node
        self.head = new_node
        print(Fore.GREEN+"\n\nInsert at start
Successfully"+Style.RESET_ALL)

    def insert_at_end(self, data): # Inserting a node at the end of the
list
        new_node = Node(data)
        if self.size() == 0: # Checking if the list is empty
            self.head = new_node # If the list is empty, just assign the
new node to the head node
            return print(Fore.GREEN + "\n\nData inserted at head
instead"+ Style.RESET_ALL)
        temp = self.head
        while temp.next is not None: # Traversing the list to find the
last node then assign the new node to the last node's next node
            temp = temp.next
        temp.next = new_node
        print(Fore.GREEN+ "\n\nData inserted successfully at the tail" +
Style.RESET_ALL)

    def insert_at_mid(self, data, position): # Inserting a node at the
middle of the list with the given index/position
        if position == 0: # If the given index/position is 0, just assign
the new node to the head node
            return self.insert_at_start(data)

        new_node = Node(data) # Creating a new node
        ptr = None
        temp = self.head
```



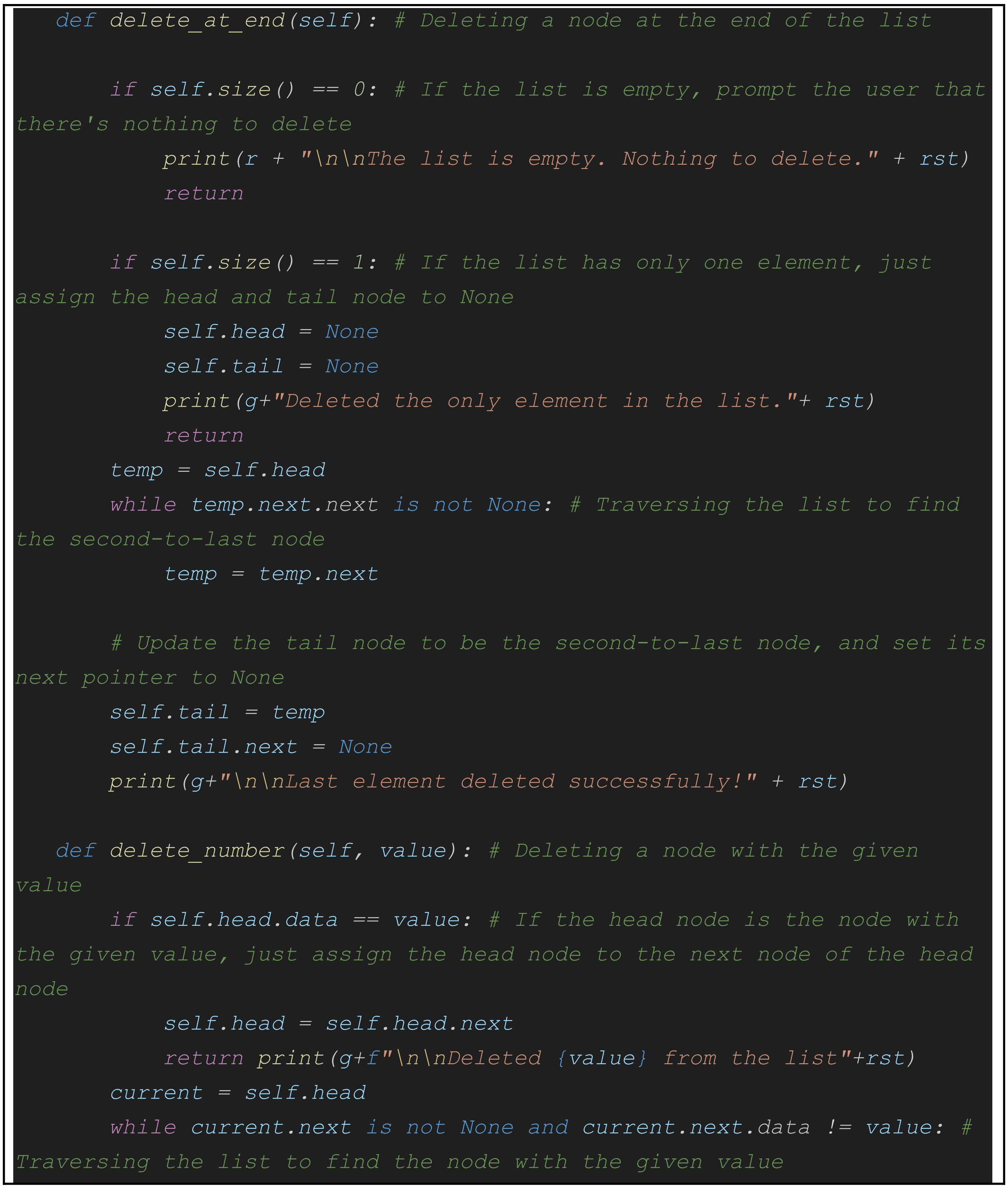
```
        count = 0
        if position < 0: # If the given index/position is negative we
enter this condition
            if position < (self.size() - (self.size()*2)): # If the given
index/position is less than the negative size of the list, it is out of
range
                print(Fore.RED+"\n\nIndex out of range" +
Style.RESET_ALL)
                return
            if position == (self.size() - (self.size()*2)): # If the
given index/position is equal to
                                                            # the
negative size of the list, just assign the new node to the head node
                return self.insert_at_start(data)
            position += self.size() + 1 # If the given index/position is
greater than the negative size of the list, convert it to positive
            while count < position: # then traverse the list to find the
node with the given index/position
                ptr = temp
                temp = temp.next
                count += 1
            new_node.next = temp
            ptr.next = new_node
            position -= self.size() # after assigning the new node to the
node with the given index/position, convert the index/position back to
negative
            print(Fore.GREEN + f"\n\nSuccessfully Inserted at Index
{position}."+ Style.RESET_ALL) # prompt the user that the new node is
successfully inserted at the given index/position
            return
        if position > self.size(): # If the given index/position is
greater than the size of the list, it is out of range
            print(Fore.RED+"\n\nIndex out of range" + Style.RESET_ALL)
            return
```



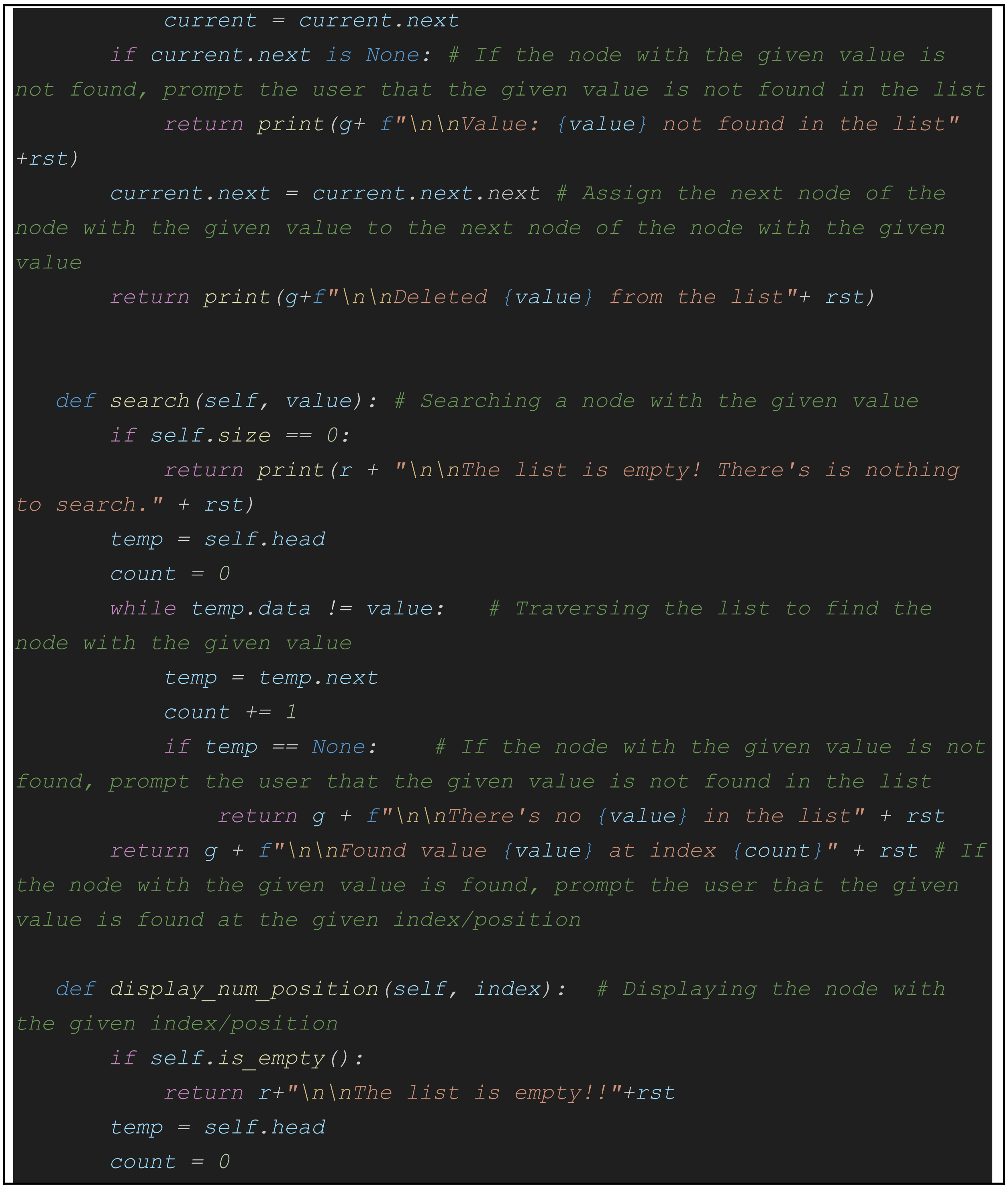
```
       while count < position: # If the given index/position is within
the range, traverse the list to find the node with the given
index/position
           ptr = temp
           temp = temp.next
           count += 1 # after this loop ptr will be the node with the
given index/position and temp will be the next node of ptr
       new_node.next = temp
       ptr.next = new_node
       print(Fore.GREEN + f"\n\nSuccessfully Inserted at Index
{position}." + Style.RESET_ALL) # prompt the user that the new node is
successfully inserted at the given index/position
   def delete_at_start(self): # Deleting a node at the start of the
list, just assign the head node to the next node of the head node
       self.head = self.head.next
       return print(g +"\n\nDelete at start successfully" + rst)

   def delete_at_mid(self, position): # Deleting a node at the middle of
the list with the given index/position
       if position == 0: # If the given index/position is 0, just assign
the new node to the head node
           return self.delete_at_start()
       ptr = None
       temp = self.head
       count = 0
       if position < 0: # If the given index/position is negative we
enter this condition
           if position < (self.size() - (self.size()*2)): # If the given
index/position is less than the negative size of the list, it is out of
range
               print(r+ "\n\nIndex out of range"+ rst)
               return
```



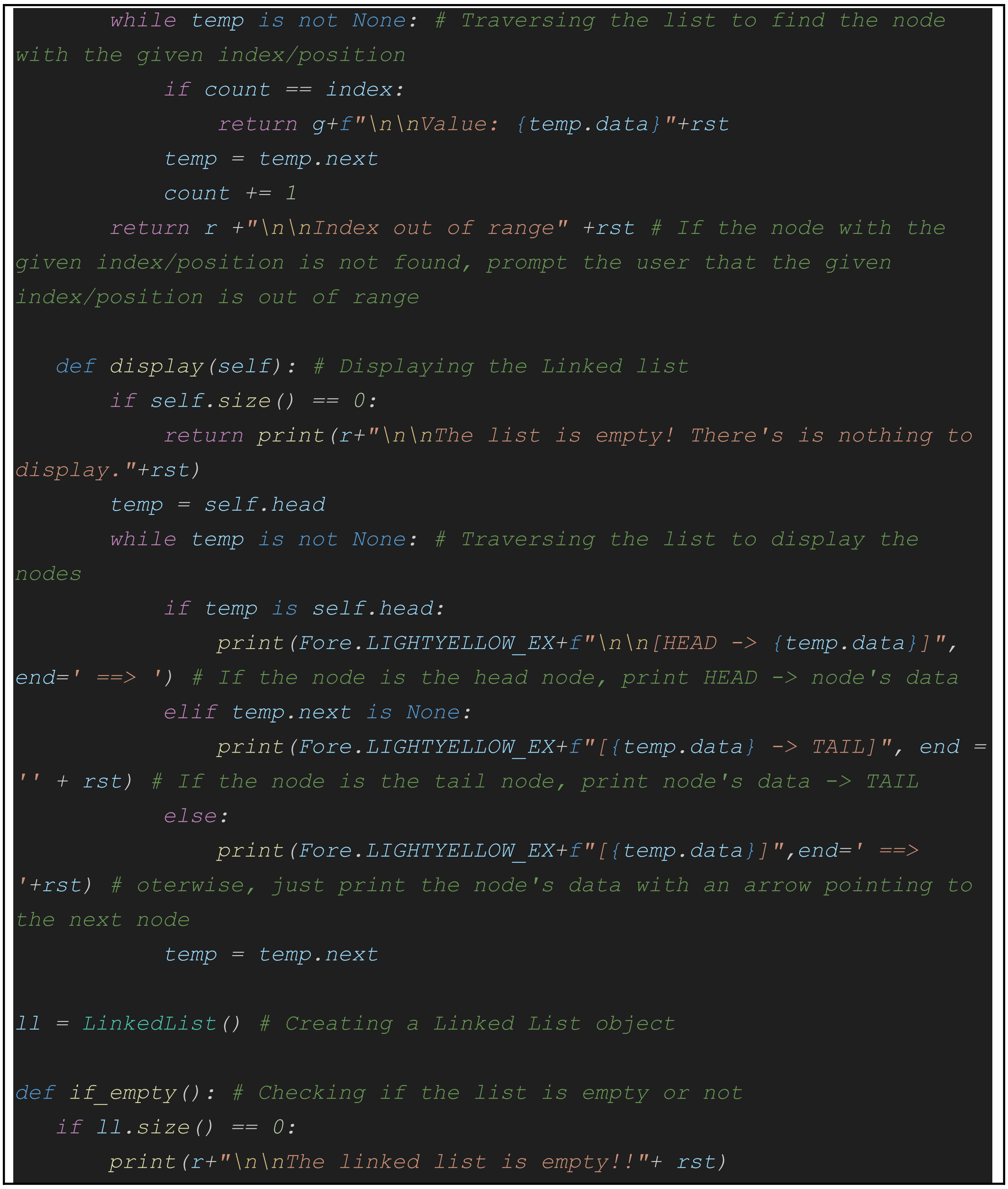
```
            if position == (self.size() - (self.size()*2)): # If the
given index/position is equal to
                                                           # the
negative size of the list, just assign the new node to the head node
                return self.delete_at_start()
            position += self.size() # If the given index/position is
greater than the negative size of the list, convert it to positive
            while count < position: # then traverse the list to find the
node with the given index/position
                ptr = temp
                temp = temp.next
                count += 1
            ptr.next = temp.next
            position -= self.size()+1# after assigning the new node to
the node with the given index/position, convert the index/position back
to negative
            print(g + f"\n\nSuccessfully Deleted at index {position}." +
rst) # prompt the user that the new node is successfully inserted at the
given index/position
            return
        if position > self.size() -1: # If the given index/position is
greater than the size of the list, it is out of range
            return print(r +"\n\nIndex out of range" + rst)
        while count < position: # If the given index/position is within
the range, traverse the list to find the node with the given
index/position
            ptr = temp
            temp = temp.next
            count += 1 # after this loop ptr will be the node with the
given index/position and temp will be the next node of ptr
        ptr.next = temp.next
        print(g +f"\n\nSuccessfully Deleted at Index {position}." + rst)
# prompt the user that the node with the given index/position is
successfully deleted
```



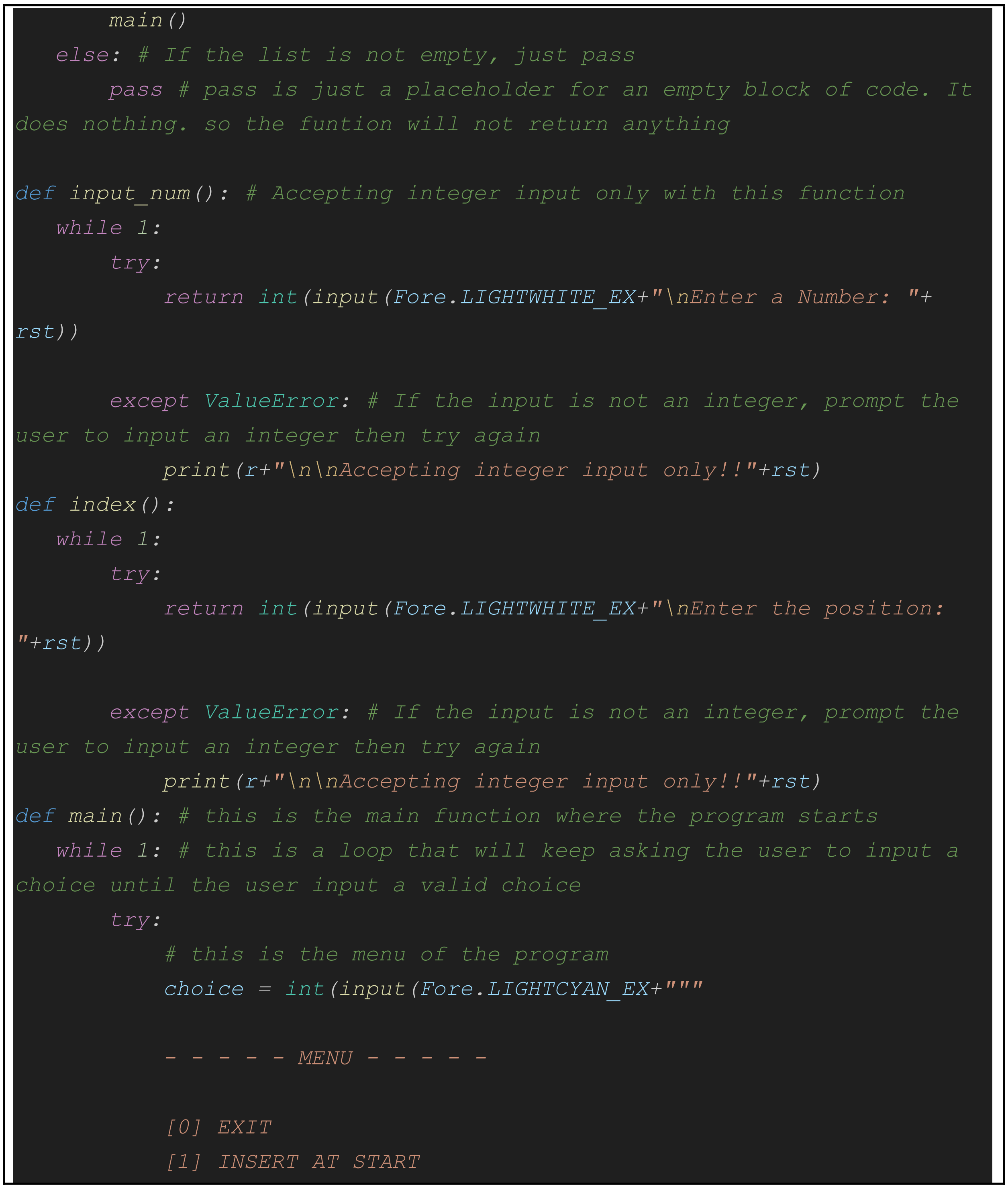
```
    def delete_at_end(self): # Deleting a node at the end of the list

        if self.size() == 0: # If the list is empty, prompt the user that there's nothing to delete
            print(r + "\n\nThe list is empty. Nothing to delete." + rst)
            return

        if self.size() == 1: # If the list has only one element, just assign the head and tail node to None
            self.head = None
            self.tail = None
            print(g+"Deleted the only element in the list."+ rst)
            return
        temp = self.head
        while temp.next.next is not None: # Traversing the list to find the second-to-last node
            temp = temp.next

        # Update the tail node to be the second-to-last node, and set its next pointer to None
        self.tail = temp
        self.tail.next = None
        print(g+"\n\nLast element deleted successfully!" + rst)

    def delete_number(self, value): # Deleting a node with the given value
        if self.head.data == value: # If the head node is the node with the given value, just assign the head node to the next node of the head node
            self.head = self.head.next
            return print(g+f"\n\nDeleted {value} from the list"+rst)
        current = self.head
        while current.next is not None and current.next.data != value: # Traversing the list to find the node with the given value
```



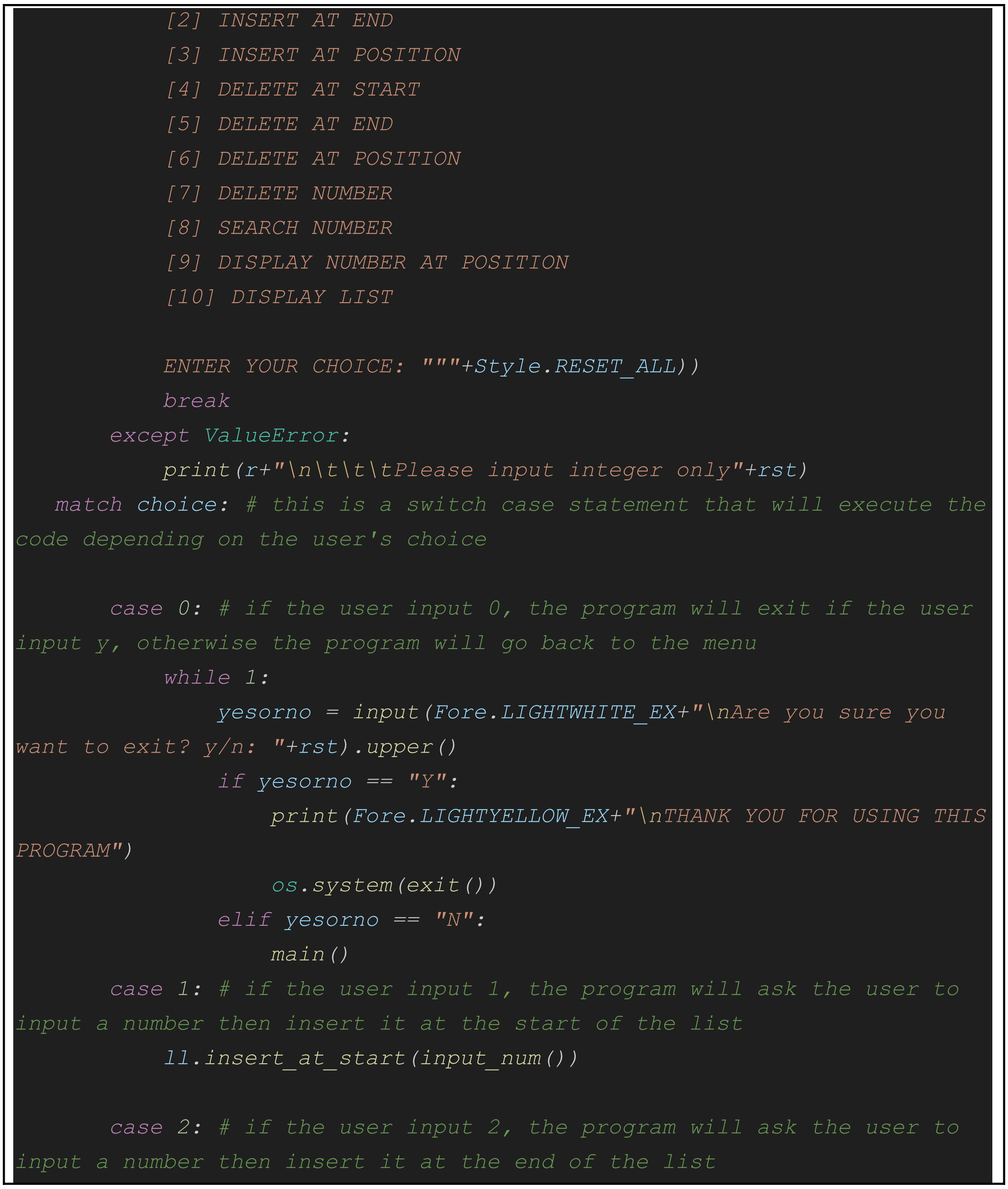
```
            current = current.next
        if current.next is None: # If the node with the given value is
not found, prompt the user that the given value is not found in the list
            return print(g+ f"\n\nValue: {value} not found in the list"
+rst)
        current.next = current.next.next # Assign the next node of the
node with the given value to the next node of the node with the given
value
        return print(g+f"\n\nDeleted {value} from the list"+ rst)


    def search(self, value): # Searching a node with the given value
        if self.size == 0:
            return print(r + "\n\nThe list is empty! There's is nothing
to search." + rst)
        temp = self.head
        count = 0
        while temp.data != value:   # Traversing the list to find the
node with the given value
            temp = temp.next
            count += 1
            if temp == None:    # If the node with the given value is not
found, prompt the user that the given value is not found in the list
                return g + f"\n\nThere's no {value} in the list" + rst
        return g + f"\n\nFound value {value} at index {count}" + rst # If
the node with the given value is found, prompt the user that the given
value is found at the given index/position

    def display_num_position(self, index):  # Displaying the node with
the given index/position
        if self.is_empty():
            return r+"\n\nThe list is empty!!"+rst
        temp = self.head
        count = 0
```



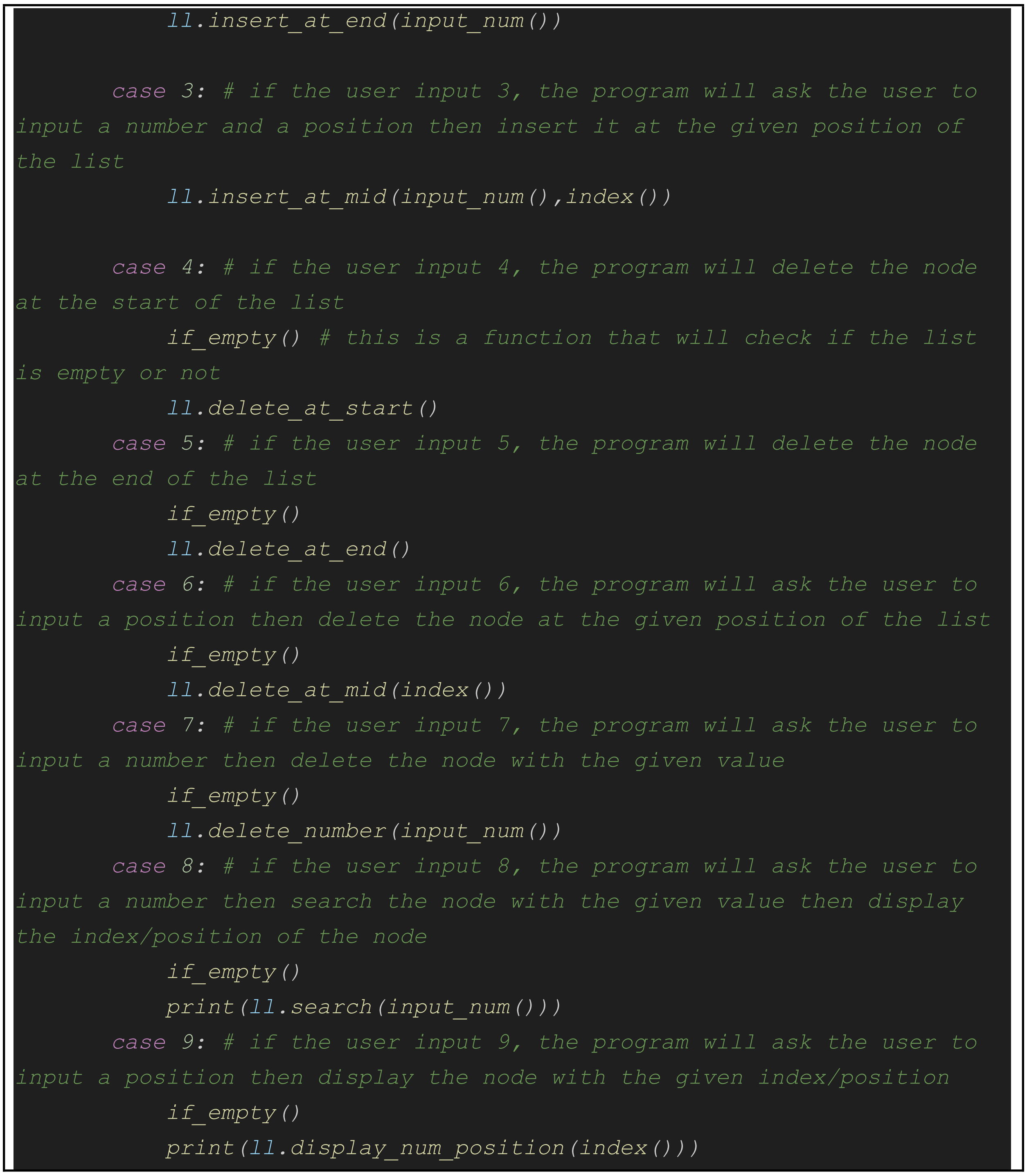
```
        while temp is not None: # Traversing the list to find the node with the given index/position
            if count == index:
                return g+f"\n\nValue: {temp.data}"+rst
            temp = temp.next
            count += 1
        return r +"\n\nIndex out of range" +rst # If the node with the given index/position is not found, prompt the user that the given index/position is out of range

    def display(self): # Displaying the Linked list
        if self.size() == 0:
            return print(r+"\n\nThe list is empty! There's is nothing to display."+rst)
        temp = self.head
        while temp is not None: # Traversing the list to display the nodes
            if temp is self.head:
                print(Fore.LIGHTYELLOW_EX+f"\n\n[HEAD -> {temp.data}]", end=' ==> ') # If the node is the head node, print HEAD -> node's data
            elif temp.next is None:
                print(Fore.LIGHTYELLOW_EX+f"[{temp.data} -> TAIL]", end = '' + rst) # If the node is the tail node, print node's data -> TAIL
            else:
                print(Fore.LIGHTYELLOW_EX+f"[{temp.data}]",end=' ==> '+rst) # oterwise, just print the node's data with an arrow pointing to the next node
            temp = temp.next

ll = LinkedList() # Creating a Linked List object

def if_empty(): # Checking if the list is empty or not
    if ll.size() == 0:
        print(r+"\n\nThe linked list is empty!!"+ rst)
```



```
        main()
    else: # If the list is not empty, just pass
        pass # pass is just a placeholder for an empty block of code. It
does nothing. so the funtion will not return anything

def input_num(): # Accepting integer input only with this function
    while 1:
        try:
            return int(input(Fore.LIGHTWHITE_EX+"\nEnter a Number: "+
rst))

        except ValueError: # If the input is not an integer, prompt the
user to input an integer then try again
            print(r+"\n\nAccepting integer input only!!"+rst)
def index():
    while 1:
        try:
            return int(input(Fore.LIGHTWHITE_EX+"\nEnter the position:
"+rst))

        except ValueError: # If the input is not an integer, prompt the
user to input an integer then try again
            print(r+"\n\nAccepting integer input only!!"+rst)
def main(): # this is the main function where the program starts
    while 1: # this is a loop that will keep asking the user to input a
choice until the user input a valid choice
        try:
            # this is the menu of the program
            choice = int(input(Fore.LIGHTCYAN_EX+"""

            - - - - - MENU - - - - -

            [0] EXIT
            [1] INSERT AT START
```



```
            [2] INSERT AT END
            [3] INSERT AT POSITION
            [4] DELETE AT START
            [5] DELETE AT END
            [6] DELETE AT POSITION
            [7] DELETE NUMBER
            [8] SEARCH NUMBER
            [9] DISPLAY NUMBER AT POSITION
            [10] DISPLAY LIST

            ENTER YOUR CHOICE: """+Style.RESET_ALL))
            break
        except ValueError:
            print(r+"\n\t\t\tPlease input integer only"+rst)
    match choice: # this is a switch case statement that will execute the
code depending on the user's choice

        case 0: # if the user input 0, the program will exit if the user
input y, otherwise the program will go back to the menu
            while 1:
                yesorno = input(Fore.LIGHTWHITE_EX+"\nAre you sure you
want to exit? y/n: "+rst).upper()
                if yesorno == "Y":
                    print(Fore.LIGHTYELLOW_EX+"\nTHANK YOU FOR USING THIS
PROGRAM")
                    os.system(exit())
                elif yesorno == "N":
                    main()
        case 1: # if the user input 1, the program will ask the user to
input a number then insert it at the start of the list
            ll.insert_at_start(input_num())

        case 2: # if the user input 2, the program will ask the user to
input a number then insert it at the end of the list
```



```
           ll.insert_at_end(input_num())

       case 3: # if the user input 3, the program will ask the user to
input a number and a position then insert it at the given position of
the list
           ll.insert_at_mid(input_num(),index())

       case 4: # if the user input 4, the program will delete the node
at the start of the list
           if_empty() # this is a function that will check if the list
is empty or not
           ll.delete_at_start()
       case 5: # if the user input 5, the program will delete the node
at the end of the list
           if_empty()
           ll.delete_at_end()
       case 6: # if the user input 6, the program will ask the user to
input a position then delete the node at the given position of the list
           if_empty()
           ll.delete_at_mid(index())
       case 7: # if the user input 7, the program will ask the user to
input a number then delete the node with the given value
           if_empty()
           ll.delete_number(input_num())
       case 8: # if the user input 8, the program will ask the user to
input a number then search the node with the given value then display
the index/position of the node
           if_empty()
           print(ll.search(input_num()))
       case 9: # if the user input 9, the program will ask the user to
input a position then display the node with the given index/position
           if_empty()
           print(ll.display_num_position(index()))
```

```
        case 10: # if the user input 10, the program will display the
list
            ll.display()
        case _:
            print(r+"\n\nInvalid Choice!!"+rst) # if the user input an
invalid choice, the program will prompt the user that the choice is
invalid then go back to the menu
            main()
    main() # this is a recursive function that will keep the program
running unless the user inputs 0 then confirm to exit the program
main() # This is to call the main function to start the program
```

TUTORIAL VIDEO

YouTube Link: https://youtu.be/xCah_eRS80A

GitHub Repository link:
https://github.com/s0y4hh/DataStructureAndAlgorithm/blob/master/Linked_List.py

TAKEAWAYS

In this learning task, I learn more about linked lists and all of the functions of it. Linked list is not just a simple list that you just code anything to be able to add,delete,search etc. you need a proper understanding on how the linked list works so it happens to me while writing this code I encounter so many bugs and errors for some test cases because not like a simple array you can't easily manipulate its values because you need to keep track of every nodes while doing the functions and also the trickiest part is the error handling code where I need to test every possible input so that I can handle all possible error while running the program. In summary I learned that a linked list is not that simple, you can modify or add some features to the code to make it a more optimized and error free program.